Why the Use of Domain-specific Modeling in Airworthy Software Requires New Methods and How these Might Look Like? (extended version)

Vanessa Tietz (vanessa.tietz@ils.uni-stuttgart.de)
University of Stuttgart
Constantin Frey (constantin.frey@ils.uni-stuttgart.de)
University of Stuttgart
Andreas Waldvogel (andreas.waldvogel@ils.uni-stuttgart.de)
University of Stuttgart
Julian Schoepf (julian.schoepf@ils.uni-stuttgart.de)
University of Stuttgart
Bjoern Annighoefer (bjoern.annighoefer@ils.uni-stuttgart.de)
University of Stuttgart

Research Article

Keywords:

DOI: https://doi.org/

Additional Declarations: No competing interests reported.

Why the Use of Domain-specific Modeling in Airworthy Software Requires New Methods and How these Might Look Like? (extended version)

Vanessa Tietz[1*], Constantin Frey[1], Andreas Waldvogel[1], Julian Schoepf[1] and Bjoern Annighoefer[1]

[1]Institute of Aircraft Systems, University of Stuttgart, Pfaffenwaldring 27, Stuttgart, 70569, BW, Germany.

*Corresponding author(s). E-mail(s): vanessa.tietz@ils.uni-stuttgart.de;
Contributing authors: constantin.frey@ils.uni-stuttgart.de; andreas.waldvogel@ils.uni-stuttgart.de; julian.schoepf@ils.uni-stuttgart.de; bjoern.annighoefer@ils.uni-stuttgart.de;

Abstract

The use of domain-specific modeling (DSM) in safety-critical avionics is rare, even though the ever-increasing complexity of avionics systems makes the use of DSM reasonable. DSM shows its advantage especially in capturing complex systems, data and relationships. The reason for the limited use in the (safety-critical) avionics area is mainly due to the high demands on the safety of software and systems. Everything that is to be used in flight operations and development must undergo a rigorous and complex certification process. Any data used in operations must be verified. A reduction of this effort can be achieved using qualified tools. A qualified tool can either replace or support certification activities. This article elaborates different use cases of how DSM could be used in relation to airworthy software. For those use cases, we review the effort of a certification and retrieve the major shortcomings and showstoppers of available frameworks, e.g. infeasible qualification of DSM runtimes and the unavailability of qualification artifacts. Finally, we elaborate possible ways of mitigation and show the concept and first results of a new DSM framework for airworthy applications, called DOMAINES. DOMAINES covers deterministic meta-modeling up to graph-based model transformations and verified visual editing. The concept of DOMAINES and a first functional prototype are presented that indicate that the shortcomings can be mitigated.

Keywords: Domain-specific modeling, safety-critical, avionics, certification, qualification

1 Introduction

The complexity of systems and software used in aircraft and the associated development effort are constantly increasing. To cope with this complexity, it is conceivable to use concepts of domain-specific modeling (DSM) in the field of avionics. DSM in general copes with the challenges of building complex and heterogeneous systems [40] and software. A particular challenge when using DSM

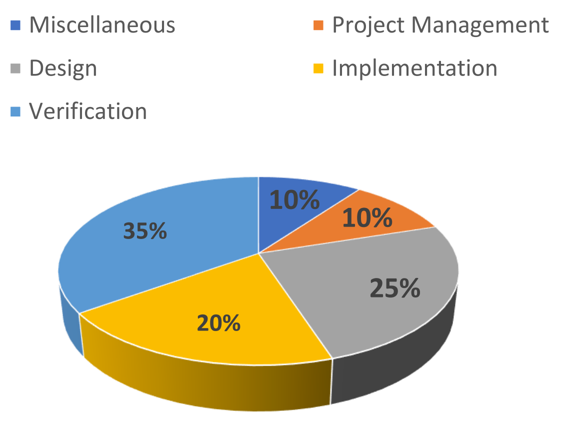

Fig. 1 Effort distribution for DAL C software [27]

in avionics is the issue of safety. Most avionics software is safety-critical and for this reason, must go through a rigorous life cycle process for certification. Both in the development process and in real-time applications, the developer must ensure that no misbehavior occurs nor that there is unacceptable negative impact on the safety of the aircraft. Acceptable evidence of freedom from misbehavior must be provided to a certification authority in the form of documents and supporting artifacts.

The process of software certification takes up a considerable part of the software development effort in the avionics domain, as depicted in Figure 1 [27]. This effort distribution shows that 35% of the effort results from certification activities for software developed according to item development assurance level (DAL) C, which is acceptable for software with the safety-effect “Major” [51]. There are different safety-effects ranging from “No Safety Effect” to “Catastrophic” depending on the influence on aircraft, crew, and passengers. DAL A software, required in case of any “Catastrophic” safety-effect, increases the absolute effort and the relative contribution of any certification activity by orders of magnitude (cf. DO-178C [46]). However, exact effort quantifications are hard to obtain.

When aiming to leverage existing DSM tools in the safety-critical domain, one possibility to use their outputs (e.g. meta-model, model formats, or source-code) is to conduct a (manual) output verification. Another theoretical possibility is to qualify existing tools according to their application. However, this usually fails because the effort is almost the same as for the flying software itself and often required documents are missing and the software foundations are infeasible for qualification. In both cases, the effort is prohibitive, which makes the use of current DSM tools end at the stage of support tools. Their outputs and benefits hardly enter the qualified region of safety-critical systems. Even more ambitious is the use of DSM methods in flying software. For this purpose DSM software must be fully certified, since manual output verification is not possible. All generated artifacts needed for further operations would first have to be manually checked for correctness before they could be processed. In [56], we already presented the basic concept for the DSM framework DOMAINES (**do**main-specific **m**odeling for **ai**rcraft a**n**d other **e**nvironment**s**), that is to be used in the safety-critical domain. The current paper aims to justify the need for the DOMAINES framework. It elaborates the reasons for the high qualification effort of current frameworks and to show how it can be reduced in order to leverage the full potential of DSM when developing safety-critical systems as well as in the execution of safety-critical (flying) real-time software. We also aim to show how DOMAINES can be developed from the proposed measures and how they interact with each other to achieve the goal of a certifiable DSM framework.

The paper is structured as follows: First basic knowledge about avionics software development, domain-specific modeling, and conceivable use cases of DSM in avionics is presented. Section 3 explains the reasons for the extensive qualification effort required when using DSM tools in avionics. In addition, section three provides suggestions on how to reduce the possible qualification effort. Our approach, as well as initial results for the development of a certifiable DSM framework, is presented in Section 4. Finally, there is a summary and an outlook. This article is an extended version of [57]. It adds more details, figures, and the current state of DOMAINES.

2 Domain-specific Modeling and (Airworthy) Safety-critical Software

This chapter focuses on the fundamentals of DSM within the domain of safety-critical avionics

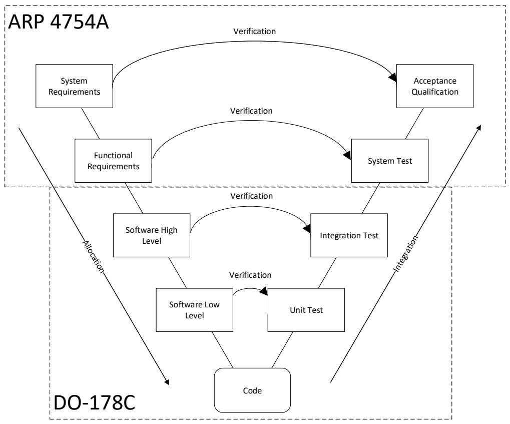

Fig. 2 Avionics software development process [51][46]

systems. It gives an overview of the development process of safety-critical avionics software with respect to its dedicated certification process, describes the fundamentals of DSM, and presents possible use cases of DSM within the field of avionics system development.

2.1 Avionics Software Development and Certification Process

Avionics software always operates as part of an overall system. Therefore, the system life cycle process is considered first. Acceptable means for showing compliance of the aircraft system development with the airworthy regulations are contained in the aviation standard ARP 4754A [51]. The ARP 4754A process model is the V-model as depicted in Figure 2. The software life cycle process is divided into a development process and a verification process, seen on the left and right branches of the V-model, respectively. On every level of the process, output data is produced in terms of certification artifacts. This data includes software high level and software low level requirements, design descriptions, and traceability data.

A software development assurance level (DAL) is assigned to the software based on the results from the safety assessment process during system development. The software level defines the objectives and activities of the software life cycle. For software with a software level DAL C, this includes, but is not limited to, the development of software high-level requirements, software architecture, software low-level requirements, and source code. Acceptable means of compliance for achieving the required software level are contained in the software standard DO-178C [46]. The DO-178C is accompanied by technique-specific supplements. These are the software tool qualification considerations DO-330 [47], the model-based development and verification supplement DO-331 [48], the object-oriented technology and related techniques supplement DO-332 [49] and the formal methods supplement DO-333 [50]. The most fundamental level of the avionics software development process is the source code. According to DO-178C, the source code for DAL C software must have the properties traceable, verifiable, consistent, and correctly implements low-level requirements [46]. The most prominent programming languages used in safety-critical software projects are C and Ada. Ravenscar Ada [12] and MISRA-C [38] describe additional restrictions for safety-critical programming.

In the verification process, the software is analyzed and tested for errors. Verification is performed for the outputs of the development process. For DAL C software the objectives of the verification process are e.g., to confirm accuracy, consistency, verifiability, and conformance to the requirements as well as conformance to the requirements on the next higher level. The output of this process is e.g., test results and traceability data.

The entire processes of the software life cycle must be repeated iteratively. Therefore, parts of the software life cycle process should be automated with the help of software tools. A software tool is defined as a type of software that is "used to help develop, transform, test, analyze, produce, or modify another program" [47]. Even if a "tool cannot introduce an error in the output of the tool, but may fail to detect an error" [47], it has to be qualified for its use. In contrast to software, tools are not certified as part of a system but receive a qualification only for their specific use. The guidelines for tool qualification are defined in the DO-330 standard [47]. The tool life cycle process is similar to the software life cycle process. According to the software life cycle process a tool qualification level (TQL), ranging from TQL-1 (most rigorous) to TQL-5 (least rigorous), has to be assigned to the tool. This is determined according to the influence of the tool on the software life cycle. It defines the objectives and activities of the tool life cycle. Comparing the number of objectives and activities to be achieved for a tool with TQL-1 and software with DAL A, the effort for the tool life cycle can be as high as for the software life cycle. Depending on the TQL, the artifacts for tool qualification at least include tool high level requirements, tool architecture description, and tool low level requirements (not for TQL-4 and TQL-5). In contrast to the software life cycle process, the tool life cycle process is divided into an operational process and a development and verification process. In the operational process, the operational requirements are defined and verified by the tool user. Furthermore, the tool user is responsible for the qualification of the tool as defined in the software life cycle. In the development and verification process, the tool requirements are defined and verified by the tool developer. A tool qualification kit is used for communication between the tool user and the tool developer. “The developer should provide [the] qualification kit/package to assist the user in the qualification process” [28].

A key property for software tool qualification is determinism as it is defined in the DO-330 [47] standard: “For a tool whose output may vary within expectations, it should be shown that the variation does not adversely affect the intended use of the output and that the correctness of the output can be established.”.However, if the output from a tool is to be used in software, the definition of determinism is stricter, in the sense that a variation of the output for the same input when operating in the same environment is not allowed [47].

A note on terminology: A distinction is made between the terms qualification and certification in the area of authorization. Qualification is used when dealing with an “auxiliary” tool and certification when dealing with software used in the aircraft. In ambiguous cases, this article uses certifiability.

The use of object-oriented programming languages and techniques is controversial in both the software life cycle and the tool life cycle. In non-critical software, object-oriented programming is widespread. In the field of avionics its use is increasing, nevertheless, several issues have to be considered to ensure safety and integrity as described in the DO-332 standard [49]. The issues described include inheritance, overloading, type conversion, exception management, and dynamic memory management. In the area of inheritance, the problems mainly concern the inheritance of methods and their implementation, especially within multiple inheritance. Despite the improvement of code readability and maintenance, overloading can induce ambiguities when the compiler performs inappropriate implicit type conversions. Many problems are induced by implicit type conversions, e.g. with a narrowing type conversion, data may be lost. A potential challenge is the exception management. Exception handling breaks the code hierarchy, is difficult to verify, and endangers determinism. Moreover, it is based on dynamic memory allocation. Since object-oriented programming tends to rely on dynamic memory management techniques, this is an important point to consider. The vulnerabilities include ambiguous references, starvation, or heap memory exhaustion. Another important point is traceability, which must also be taken into account in procedural programming, but causes a few difficulties in object-oriented programming such as constructors, and destructors complicating the traceability between object code and source code [44].

Formal methods can be used to support the validation and verification of safety-critical software. These are mathematically based techniques for the specification, development, and verification of software aspects of digital systems [50]. The usage of formal methods is motivated by mathematical analyses contributing to establishing the correctness of a design. As noted in the DO-333, formal methods are capable of providing verification evidence, demonstrating the freedom from exceptions or unintended function, and are therefore an important resource for achieving qualifiability and certifiability.

2.2 Domain-specific Modeling

DSM is typically used to cope with large and complex data structures when developing and deploying systems and software. DSM raises the level of abstraction beyond coding, making development faster and easier [32]. Several frameworks, such as the Eclipse Modeling Framework (EMF) [53], the Generic Modeling Environment (GME) [35], the Cameo Systems Modeler [15], or the Mathworks

System Composer [37], are available for DSM. These frameworks allow the efficient handling of complex relationships in application-specific data models using a meta-language, a meta-model, and a domain-specific model. Domain-specific models can be defined as an abstraction of a cyber-physical system that represents a partial and simplified view [40]. Meta-languages are the foundation of each meta-model. Typically representations of meta-languages can either be graphical (e.g.: UML [42], (E)MOF [43], MetaGME [29]) or text-based (e.g.: Backus-Naur Form (BNF) [45]). The meta-model defines a domain-specific language for the domain-specific model. Domain-specific languages are tailored to a particular application domain (e.g.: OAAM [3], AADL [20]). Furthermore, the use of domain-specific languages can help to bridge the gap between domain-experts and the implementation of modeled systems. Experts can keep their language and information can be automatically analyzed and implementations like source code can be generated [34]. DSM applications can range from highly specific (e.g. avionics) to broader domains (e.g. embedded systems) [22].

For further consideration, we refer to the meta-language as M3-level, the meta-model as the M2-level, and the domain-specific model as the M1-level.

Additionally, DSM enables an increasing level of automation, which contributes to more efficient processes and higher product maturity [40]. Automations can be realized by means of model transformations which can either be model to model (M2M), text to model (T2M), or model to text (M2T). A uniform definition of model transformation is difficult, as there are many different approaches. Basically, a model transformation transfers a source model into a target model, both can be either graphical object-based or text-based. Model transformations consist of a set of rules that define the information transfer. Rules can exist as e.g., textual statements or graphical, object-based representations. A single rule contains a description of converting a source construct into a target construct [16]. The model transformation approaches include the operational approach, relational approach, hybrid structures, and graph-based methods. Examples of these transformation approaches are contained in the Query/View/Transformation (QVT) specification of the Object Management Group (OMG) [41]. The Atlas Transformation Language (ATL) is a hybrid transformation language that combines declarative and imperative constructs [31]. Model transformations can also be realized with graph transformations. An example of this is the transformation language GReAT [8].

2.3 Domain-specific Modeling in Avionics

In avionics, there are several application areas for DSM e.g. requirements formalization [39], pre-design, simulation [26], configuration [4] [24] [52], optimization [9], code-generation and automated testing [30] [39]. We see high potentials for the use of DSM in, first, the development of flying systems and, second, as part of flying software itself. In the following section, we introduce three use cases, which we then analyze in Section 3.

2.3.1 Use Case 1: Generation of Avionic Configurations and Simulation Models

This use case describes the implementation of a seamless toolchain for the design, implementation, and simulation of Integrated Modular Avionics (IMA) platforms focusing on system function applications hosted on avionics [26]. The goal is to support the development of such IMA platforms by reducing error-prone tasks via automation (M2M or M2T) and more sophisticated representations of development artifacts (e.g. (object-based) configuration models instead of textual configuration files) and ensuring the generation of these development artifacts without errors using DSM tools. The toolchain consists of meta-models allowing the modeling of IMA avionics platform instances or dedicated configuration models. Model transformations allow transforming a platform model into a configuration model, which can be further processed into configuration files e.g., csv-formatted Interface Control Documents (ICDs) [25], [26]. For generating the platform and configuration model, EMF (ECORE) is used. Eclipse/EMF is based on Java and the tools used for model generation, Java Emitter Templates (JET) and Java Merge (JMerge) [24], are based on OOP procedures. An approach similar to Use Case 1 is developed within the German TALIA project [4].

2.3.2 Use Case 2: Generation of Source Code for Flight Operations

This use case targets an automated development process by generating executable code, requirement documents as well as test cases and test scripts for an advanced avionic platform. Within this approach, the characteristics of the avionic platform are described in a domain-specific language, which allows the modeling of a highly abstracted “input specification”. This “input specification” is processed and refined by multiple model transformations until executable code for platform management mechanisms of a safety-critical application is generated. Hereby, the generated source code must match the information that is contained in the software design documents [39]. These must be verified manually, unlike common code generators where the source code must match the model. For modeling of the avionic platform instances, GME is used; for processing the “Graph Rewriting And Transformation (GReAT)” language. The primary languages for integration are the OOP based ones, C++ and Visual Basic [35]. Additionally, GME comes with a C++ code generator for GReAT transformation models [8]. An approach similar to Use Case 2 is developed within the German RPAS23-CP project [11].

2.3.3 Use Case 3: Domain-specific Models in Flying Software

In this use case, the possibility of using DSM tools to generate and process domain-specific models in flying software is considered. The information available in the domain-specific models can be used for the configuration of avionics hardware modules at runtime. As an example, DSM is used by the platform consciousness of the Plug&Fly Avionics platform [6]. The Plug&Fly Avionics Platform is a concept for self-configuring /self-organizing avionics systems with the goal of providing adaptiveness during runtime for optimal dynamic resource utilization. The domain-specific platform consciousness is a model-driven database of the platform. In the platform consciousness, all information necessary for deriving the configuration (state, available resources, and application descriptions) of the modules is processed in a model-based way. In addition, automations are executed on the models e.g., in order to optimally allocate the tasks to the hardware devices. The occurrence of events in flight can lead to the need to adjust the models. This requires a runtime for model interaction. Safety-relevant decisions are made on the basis of the model information of the platform consciousness. Errors in the models, in the communication with the models, or in the processing of the models can lead to faulty decisions or configurations with effects on system safety with the safety-effect “Catastrophic”. For this reason, the highest software assurance level is needed to be assigned to the platform consciousness. The DSM runtime must therefore be developed for the software level DAL A. An approach similar to Use Case 3 is developed within the German PAFA-ONE project [6].

3 Rationale for the Extensive Qualification Effort

This section describes the possibilities of leveraging the described use cases in a safety-critical avionics environment and explains the corresponding effort that needs to be applied. All use cases use DSM to produce outputs for certified systems or even within certified software. Based on the use cases, we analyze how their output could be used in safety-critical flight operations.

3.1 Analysis of Qualification Methods of DSM in the Use Cases

Starting with **Use Case 1**, one possibility is to perform a *manual output verification*, where every development artifact has to be verified by hand [46]. This is probably the most expensive, but a natural and a commonly used option, as other options are currently hardly available. One would have to manually verify that the configuration of thousands of parameters of an IMA system generated with EMF is correct and does not lead to any misbehavior. Since there are typically a large number of different components in an aircraft, the manual verification effort can increase immensely.

This effort can and is in practice reduced by performing a *tool-based output verification* [47]. In contrast to manual output verification, the verification process is supported by tools automatically discovering wrong or malformed output with e.g., model-checkers, thus reducing manual effort. However, a verification tool used must be developed and qualified at great expense. Tool-based output verification can be conducted using a qualified model checker as well. Model checkers automatically check whether the created models meet specified requirements. *Tool-based output verification* would make it possible, for example, to create an ICD frequently used in avionics with EMF and subsequently use a qualified tool to automatically verify that the data it contains is correct. The subsequent verification of the output from EMF models, transformations, and conversions could be avoided if EMF and the ICD generator is a qualified tool itself. This would require a *tool qualification*. Although this is possible in theory, in practice this procedure will probably be infeasible. One problem is the object-oriented implementation of EMF with Java as described in Section 2.1. In addition, the complexity of the EMF is challenging. A lot of modeling constructs and implementations are included that are not needed for the application purpose, resulting in deactivated code. According to the DO-300 [47] standard, a mechanism should be designed and implemented "to provide evidence that the unexercised Tool Source Code has no impact on the outputs of the tool in any operational use.". Despite the fact that EMF is open source and the code would be customizable, the effort to ensure this is immense. Moreover, requirements would have to be created manually for any functionality within EMF.

In **Use Case 2**, source code is to be generated from the created models, which is to be used in flight operations. Again, *manual output verification* can be used to qualify the generated source code to be used in the aircraft. In doing so, the correct and intended behavior must be manually verified. Especially for DAL A software, modified condition/ decision coverage (MCDC) must be achieved to verify the source code. The problem is that the source code can be arbitrarily complex and manual verification of the output quickly reaches its limits. Depending on how far-reaching the *tool-based output verification* is, this is more or less helpful. A simple check of the source code for its syntax is possible without further ado, however, it is insufficient. It is also necessary to check whether created models have also been correctly transformed into source code, this is almost impossible. The reason for this is that the verification tool would have to understand the model and the mapping from the source code back to the model may not be explicit. The *manual qualification of a (model transformation) tool* would considerably reduce the subsequent certification effort here as well. However, the domain-specific modeling and transformation languages GME/GReAT used are hardly qualifiable. They are based on the object-oriented programming language C++. Furthermore, the transformation language GReAT does not fulfill our requirements on deterministic behavior, because depending on how the transformations are executed, different paths are taken to the output for the same input. We have also identified the assignment of random IDs for modeling objects as a critical aspect for tool qualification. Due to the random assignment of IDs, the transformation code must be completely re-qualified each time it is generated. The re-qualification is necessary because the traceability of the objects in the model is lost. Since existing domain-specific model transformation languages known to us have not been developed with regard to tool qualification for safety-critical software, we consider that model transformation tools for Use Case 2 can only be qualified with an immense amount of effort.

For **Use Case 3**, both *manual output verification* and *tool-based output verification* are impossible because the output cannot be verified at runtime. If, for example, optimization calculations were performed using a current DSM framework, these would not be allowed to be used further, as it cannot be ensured that these calculations were performed correctly. Therefore, in this case, only a *certified DSM runtime environment* comes into consideration to make DSM with its advantages usable in the aircraft. However, such a certified runtime environment does not yet exist.

3.2 Summary of the Qualification Effort of the Use Cases

Figure 3 summarizes the different methods on how to use DSM output within airworthy applications and its relation to the use cases described above. In section ① the *manual output verification* is visualized, which is adaptable to Use Case 1 and Use Case 2. For both use cases, the certification effort is extremely high, as every generated artifact, be it a configuration data sheet or source code, must be manually verified. Section ② illustrates the *tool-based output verification* where the *manual output verification* (section ①) is supported by means of already qualified tools such as model-checkers. This reduces the subsequent certification effort compared to *manual output verification*. Additional effort is required in this case due to the necessary qualification of the auxiliary tool, but this effort is nonrecurring. Section ③ shows the ideal scenario for the use of DSM in the development process for avionics systems (Use Case 1, Use Case 2), the *qualified tool*. There is no more subsequent certification effort, generated configuration files or even source code can be used directly for aircraft development. Section ④ shows Use Case 3 where output verification can not be used and therefore a certified DSM runtime is required. The major problem here, however, is that existing DSM tools can hardly be certified, which the next section will prove.

As it can be seen in Table 1, the simplest use case for DSM in the avionics area, Use Case 1, can already be applied with existing methods in the safety-critical environment, but with high effort and partly supported by qualified tools. For Use Case 2, as described, the effort required for all methods is still too high for widespread use, and for Use Case 3, there are currently no possibilities for implementing it with DSM. In all three use cases described, the benefit of a qualified DSM tool becomes clear. This may be because of the required cost-intensive output verification or simply because, as in Use Case 3, there is no DSM tool that is suitable for this purpose. Nevertheless, the effort for a tool qualification (Use Case 1, Use Case 2) and software qualification (Use Case 3) of existing DSM tools is currently still impractical.

3.3 The Hurdles for DSM Tools

In this section, the reasons for this are listed and suggestions are provided as to how this effort can be reduced. In our opinion, it is of great support to use qualified DSM tools when using DSM in (safety-critical) avionics. Unfortunately, the effort to qualify or even certify existing DSM tools seems immense.

First of all, the correct implementation of the DSM tools must comply with the guidelines for safety-critical programming. Since most DSM tools are based on **object-oriented programming languages and techniques**, such as EMF on Java, and GME on C++, proving correctness is much more difficult and costly than if they were based on a procedural language [54] as described in Section 2.1. Factors that matter in this context include traceability, (multiple) inheritance, and dynamic memory management. Software implemented with Java might be certified for a DAL D project, but DAL A software requires a higher degree of determinism [44]. Since we aim for DAL A, it seems unfeasible to use EMF for Use Case 3 at all. Moreover, it seems hard to certify software with dynamic memory allocation [44] which is required when using pointers.
Another aspect, which immensely increases the qualification effort is the **lack of qualification documents**. Although a manual or online documentation is usually provided, which could count as a design document in the broadest sense, however, this is far from being sufficient. Assuming that the documentation of existing open-source tools has a sufficiently large scope or, for example, the open source code could be analyzed, it is possible to generate the documents subsequently under high effort. Unfortunately, this would mean a reverse requirements engineering and this is not the way to fulfill the objectives from the tool standard.

To qualify a tool or certify a software any possible functionality must be checked, even if it is not required for use. For example, the behavior of the EOperations of EMF Ecore must be verified even if they are not needed for modeling configuration artifacts as in Use-Case 1. Since most DSM tools cover a large number of different use cases, have a **high level of complexity**, and thus usually offer significantly more functionality than is needed, the qualification effort is significantly higher than it

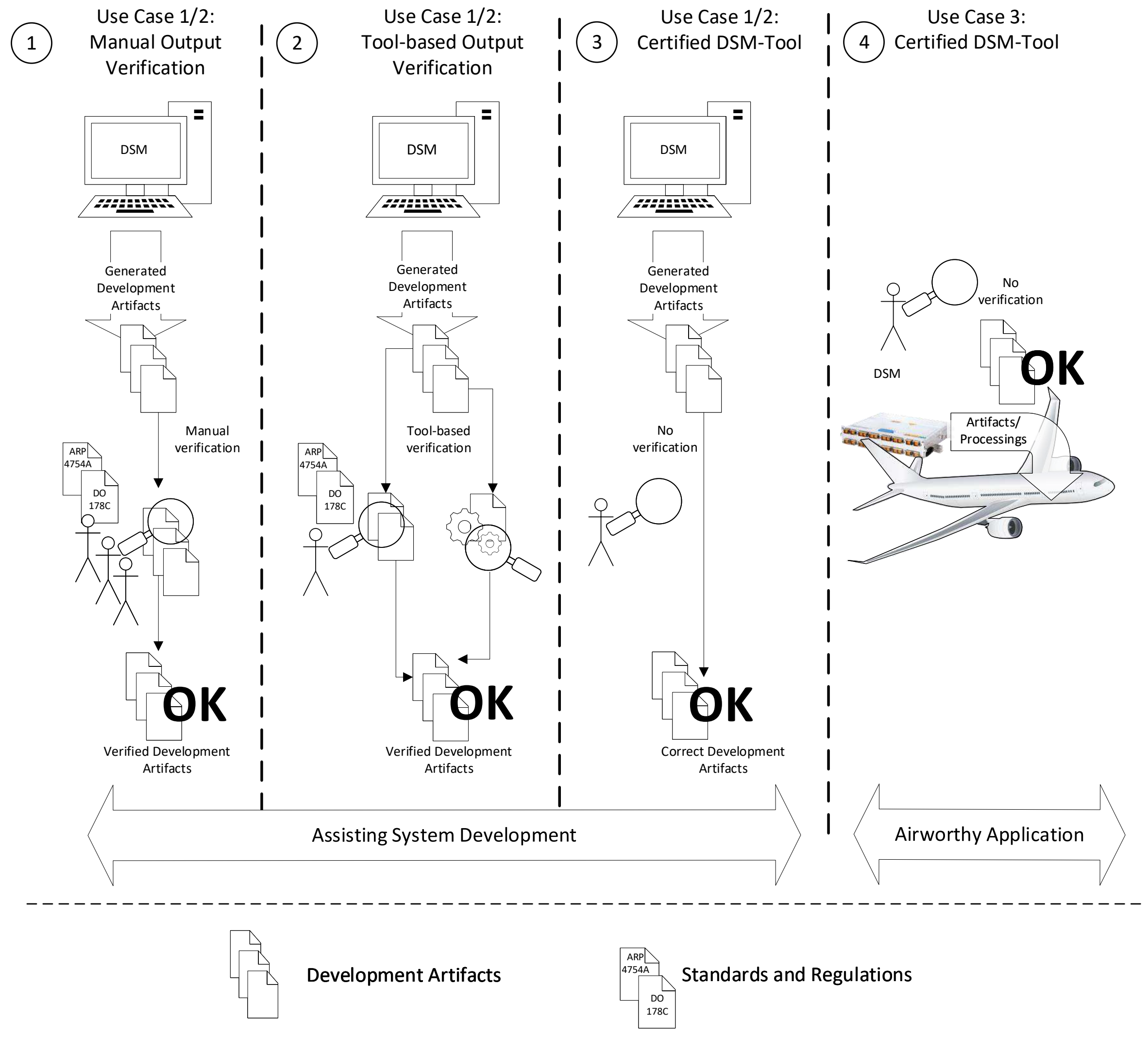
1
Use Case 1/2: Manual Output Verification
2
Use Case 1/2: Tool-based Output Verification
3
Use Case 1/2: Certified DSM-Tool
4
Use Case 3: Certified DSM-Tool
DSM
DSM
DSM
Generated Development Artifacts
Generated Development Artifacts
Generated Development Artifacts
Manual verification
Tool-based verification
No verification
No verification
DSM
OK
Artifacts/ Processings
ARP 4754A
DO 178C
ARP 4754A
DO 178C
OK
Verified Development Artifacts
OK
Verified Development Artifacts
OK
Correct Development Artifacts
Assisting System Development
Airworthy Application
Development Artifacts
ARP 4754A
DO 178C
Standards and Regulations

Fig. 3 Overview on verification methods and avionics use cases

could actually be. There is the additional issue of deactivated code. Any unused functionality could be considered deactivated code and therefore must be handled with additional measures.

A major factor for the qualification effort is the proof of **deterministic behavior** for, e.g, the execution of transformations and the creation of models. In the paper [2], the central properties of model transformations are presented and analyzed. One of these properties is determinism. In this paper, determinism for model transformations means that the same output model must always be produced for the same input model. For our understanding of determinism, further properties for model transformations from [2] include termination, traceability, and type correctness. Termination means that a transformation must be executed in finite time. To prove that the transformations and the implementation behave deterministically at any time, each transformation as well as the complete implementation must be traceable. This includes the traceability of the individual transformation steps as well as to the low-level requirements. Moreover, it must be proved that the models are type-correct with the underlying meta-language and meta-model at all times.

Table 1 Overview of use cases, verification methods and corresponding effort

Method	Use Case 1	Use Case 2	Use Case 3
Manual Output Verification	high	high	not possible
Tool-based Output Verification	medium	high	not possible
Qualified Tool	low	low	low

GReAT from GME, as used in Use-Case 2, is an example of a model transformation approach that does not meet our goals for determinism and traceability as described in 2.1. GReAT offers two ways to perform transformations [8]. Depending on whether the GReAT engine or the code generator is used for execution, the same input model can lead to a different output model or a transformation abort. This is especially the case when deleting objects from the model. In addition to verifying the created transformations, there is also the need to verify the generation of the code from existing models. This is often conducted based on OOP-based code generators.

As described, there are several factors contributing to a high qualification effort for existing DSM tools. In the following, we will examine the which methods can be used to reduce the certification effort in order to be able to develop a new certifiable DSM framework.

3.4 Suggestions on How to Reduce the Qualification Effort

The first aspect to reduce the certification effort considers the implementation of the DSM framework or generated code. A language suitable for **safety-critical programming** should be used, which, if possible, is not based on object-orientation, which seems a contradiction, because DSM originates from OOP, but also non-OOP languages can deal with objects. The standard languages suggested for safety-critical applications in avionics are Ada and C [44]. In addition to the programming language, a corresponding subset or coding guideline is suggested. MISRA-C for C and SPARK for Ada are the most prominent candidates. Moreover, SPARK provides support for formal code checking and therefore reduces the effort for source code verification. Even if the programming language is able to work in an object-oriented way, it should be avoided when implementing and managing models and, for example, the concept of object-relational mapping should be applied.

One point that can be applied at the implementation level as well as at the software architecture level to reduce the certification effort is **decoupling and modularity**. At the software architecture level, modularity can be used to separate different functional modules or components e.g., a module responsible for the visualization, model-checkers, or compilers. Each of such modules or components might have a different effect on the software and system safety and can therefore be considered with different (lower) certification levels. Moreover, if these components or modules exhibit clearly defined interfaces, decoupled incremental certification of modules becomes possible, e.g. models relying on interfaces of other modules can take certification credits on previously certified modules if ensuring that the interface is used correctly. On implementation level decoupling and modularity are primarily important when considering maintainability [44]. Readability through, among other things, reusable functions, coding guidelines, and documentation as well as well-structured code are important.

We further suggest to support the DSM tool user in the tool qualification process by offering a **tool qualification kit**. This tool qualification kit should contain relevant documents like operational, development, and verification artifacts as listed in Section 2.1. To further decrease the qualification effort, it is suggested to create the tool qualification kit in a model-based and automated way.

In addition, a qualifiable DSM framework should offer **model restrictions**. With restrictions, the effort for the verification activities might be reduced, because the number of model variants to be tested is reduced. Moreover, restrictions are capable of facilitating traceability. Restrictions can play a role in several areas, for example in the design of different modeling languages or in the development and execution of transformations. The objective is to ensure that only intended objects and relationships can be modeled. Furthermore, for transformations, restrictions can be applied either for the execution of a single transformation rule or for the rule-scheduling. Restrictive rule-scheduling ensures that the rule to be executed and its input and output are explicitly and validly defined. Restrictions on rule execution ensure that the output of the rule is always defined.

As described in Section 2.1, **formal methods** can strongly support providing verification evidences and demonstrating the freedom from unintended functionality. Therefore, formal methods are important when considering the reduction of the qualification effort. For example, they can be used to automatically verify the syntactic correctness of the source code. Formal methods can further be used for describing and analyzing rule executions of model transformations. For example, a helpful mathematical technique is graph rewriting. Moreover, formal definitions of models, languages, and restrictions, can conceivably serve as certification artifacts.

A further possibility to reduce the qualification effort is the use of **pre-qualified components** such as compilers or model-checkers.

An effective way to reduce the qualification effort is the implementation by means of **simple and comprehensible algorithms and methods**. This means that it can be helpful to forgo powerful and more complex algorithms in favor of less powerful but simple algorithms. Furthermore, special features of programming languages can also lead to a better comprehension of the implementation.

An overview of the proposed measures for reducing the certification effort and explicit suggestions are given in Table 2.

In conclusion, the effort for the certification and qualification of existing DSM frameworks is too high due to e.g., the object-oriented implementation, missing artifacts, complex (modeling and transformation) languages and software structures, and insufficient deterministic behavior. For this reason, the certifiable framework DOMAINES for DSM is being developed in the TALIA project [56]. The next chapter will discuss which of the mentioned methods we will use to achieve the lowest possible qualification effort. Moreover, the current status of DOMAINES is presented.

4 The DOMAINES Approach

To our knowledge, there are no meta-modeling tools available so far for efficient use in safety-critical software, therefore, we decided to implement and design our own DSM framework in consideration of the findings on qualifiability and certifiability. DOMAINES stands for a framework that can be used for the development of aircraft systems and in the aircraft for complex data handling and management during flight. The functional scope of existing DSM tools should not and cannot be covered - the more functionality the higher the certification effort. The next sections present requirements, the concept, and the architecture of DOMAINES, and elaborate the measures that reduce the certification effort. This description is a continuation of the DOMAINES concepts presented in [56].

4.1 Requirements for the DOMAINES Framework

For the specification of the components and the architecture of DOMAINES, requirements for its functional scope were defined. These are listed and justified below:

- **Req1:** Allow creating, using and modifying meta-models (M2) and domain-specific models (M1)
  - Rational: Enable a more efficient development, design and management of avionics systems under consideration of concepts from the model-based systems engineering.
- **Req2:** Allow graphical editing of meta-models (M2) and domain-specific models (M1)

Table 2 Overview of proposed measures to reduce the qualification effort

	Measure	Description
1	Implementation with a programming language suitable for safety-critical programming and avoidance of OOP	C with MISRA-C or Ada, usage of object relational mapping
2	Decoupling & modularity	Separate functional entities, make them interchangeable
3	Providing tool qualification kit	Qualification artifacts: requirements documents, verification documents, traceability data
4	Model restrictions	Multi-level modeling, simplified languages, augmentation with constraints, restricted rule scheduling
5	Formal methods to verify the correctness of implementations	SPARK, graph rewriting
6	Usage of pre-qualified components	Compilers, model checkers
7	Simple and comprehensible implementations and methodologies	Forgo complex algorithms in favor of simpler ones

  - Rational: Graphical editors are more intuitive for the user, provide a better overview and are usable for non-experts.

- **Req3:** Allow the verification of the visualization of meta-models (M2) and domain-specific models (M1).
  - Rational: Errors can be introduced into the system by incorrect or ambiguous representations. This is to be avoided.
- **Req4:** Allow the generation of qualification artifacts to support the tool-user.
  - Rational: The tool user is responsible for the qualification of the tool and requires suitable qualification artifacts.
- **Req5:** Allow the modeling of purely static behavior of complex relationships.
  - Rational: For our application areas within avionics, purely static modeling without the possibility to create functions or operations is sufficient.
- **Req6:** Allow the instantiation of domain-specific tools for custom purposes
  - Rational: Not every part of the DOMAINES framework is needed or useful for every application area. Therefore it should be possible to combine the modules independently of each other.
- **Req7:** Allow the execution of model-to-model (M2M), and model-to-text (M2T) transformations
  - Rational: It should be possible to apply methods of model-driven engineering to domain-specific models. This includes the generation of development and verification artifacts.
- **Req8:** Allow the persistent storage of M2 and M1 models independent of the (meta) modeling language.
  - Rational: Limit the functionality in the core of the framework to the essentials and achieve easy usability of other modeling languages.

4.2 Overview on the DOMAINES Framework

Based on the defined requirements, the following architecture and functional scope were chosen.

DOMAINES exhibits a modular hierarchic architecture. Each component contributes to the issues of qualifiability and certifiability. The high level architecture of DOMAINES is depicted in Figure 4. The red ellipses are interfaces between the components, the white boxes the components, the gray boxes an implemented language or blueprint, and the green box the elements which are specific to the modeling-language.

The central part of DOMAINES is the **MOD**eling Language (MOD) which is implemented in the **RUN**time **M**odel**D**ata**B**ase (RUNMDB). Unlike many other modeling languages, in our case, there is no text-based (xml) representation that is transformed into executable code, since it is infeasible to qualify xml parsers. Instead, the language is written directly in code.

This code is the RUNMDB, which is the executable for creating and maintaining meta-models (M2) and domain-specific models (M1). It is implemented with the programming language Ada which was especially developed for the programming of safety-critical applications and allows for formal proves with the subset SPARK. MOD and RUNMDB enable the creation of meta-models and domain-specific models as demanded by Req1.

Any interaction between the RUNMDB and the **B**asic **M**odel **I**nteraction (BMI) is performed by model-oriented CRUD (**C**reate, **R**ead, **U**pdate and **D**elete) commands. For each different RUNMDB, all that is required is a compatible language specific implementation of the BMI interface translating generic CRUD commands to more (language specific) CRUD commands. The BMI is part of the model-interaction language EOQ3 (Essential Object Query) [5] formalism.

Additionally, the **M**odel **I**nteraction **P**rocessor (MIP) is a specific implementation of the EOQ3 formalism, which is currently available in Python and (with reduced functionality) in Ada, and remains unchanged for every RUNMDB. The MIP is responsible for the translation of complex and nested queries to generic CRUD commands. By applying a model interaction language, a clear separation between model data storage and model interaction should be achieved within the framework.

While the BMI is responsible for the communication between the RUNMDB and the MIP, the **A**dvanced **M**odel **Interaction** (AMI) interface is responsible for the handling of EOQ3 commands, queries, events, and different (user) sessions for multi-user support. The formal defined AMI interface enables the efficient expansion of DOMAINES with further functionality.

Beside the creation of domain-specific languages and models, the framework provides the possibility of conducting model transformations with the Model **TRA**nsformation Engine (TRA). TRA is based on the **MetaTRA**nsformation Language (MetaTRA) which is a domain-specific language for the creation and execution of deterministic model-to-model (M2M) and model-to-text (M2T) transformations demanded by Req7.

A major difference from other DSM frameworks is the **T**ool **Q**ualification **A**gent (TQA) which is responsible for the generation of documents and artifacts required from tool users and authorities. TQA is built on TRA which has the possibility to conduct M2T transformations for document generation.

Since the RUNMDB is an empty DSM runtime, a **M**odel **P**ersistency **L**ayer (MPL) is required to persistently store and load models and meta-models via AMI commands, e.g. from files. Since MPL is interfaced to the AMI, the MPL is model and language independent addressing Req8.

The **G**raphical **ED**iting (GED) is required for the model-based visualization and graphical editing of models. This fulfills Req2.

A further novel component is the **V**isualization **V**erification **T**ool (VVT). With the VVT it shall be possible to verify if the representation of models in the GED to the user of the tool corresponds to the models in the RUNMDB, addressing Req3.

Due to the formal defined interfaces, every implementation is interchangeable. A DSM tool instantiated with DOMAINES consists of the components RUNMDB, MIP, MPL, GED and VVT including one or more model transformations. Each DSM tool must be qualified for itself. The artifacts of the **T**ool **Q**ualification **A**gent support this activity.

Any component of our framework has to cope with the issue of qualifiability and certifiability. The next section describes which measures of Section 3.4 are (to be) applied. In the following we differentiate between two different types of measures. First, measures for the effort reduction by methods and technology which are partly common in existing frameworks e.g., providing a tool qualification kit or the application of restrictions.

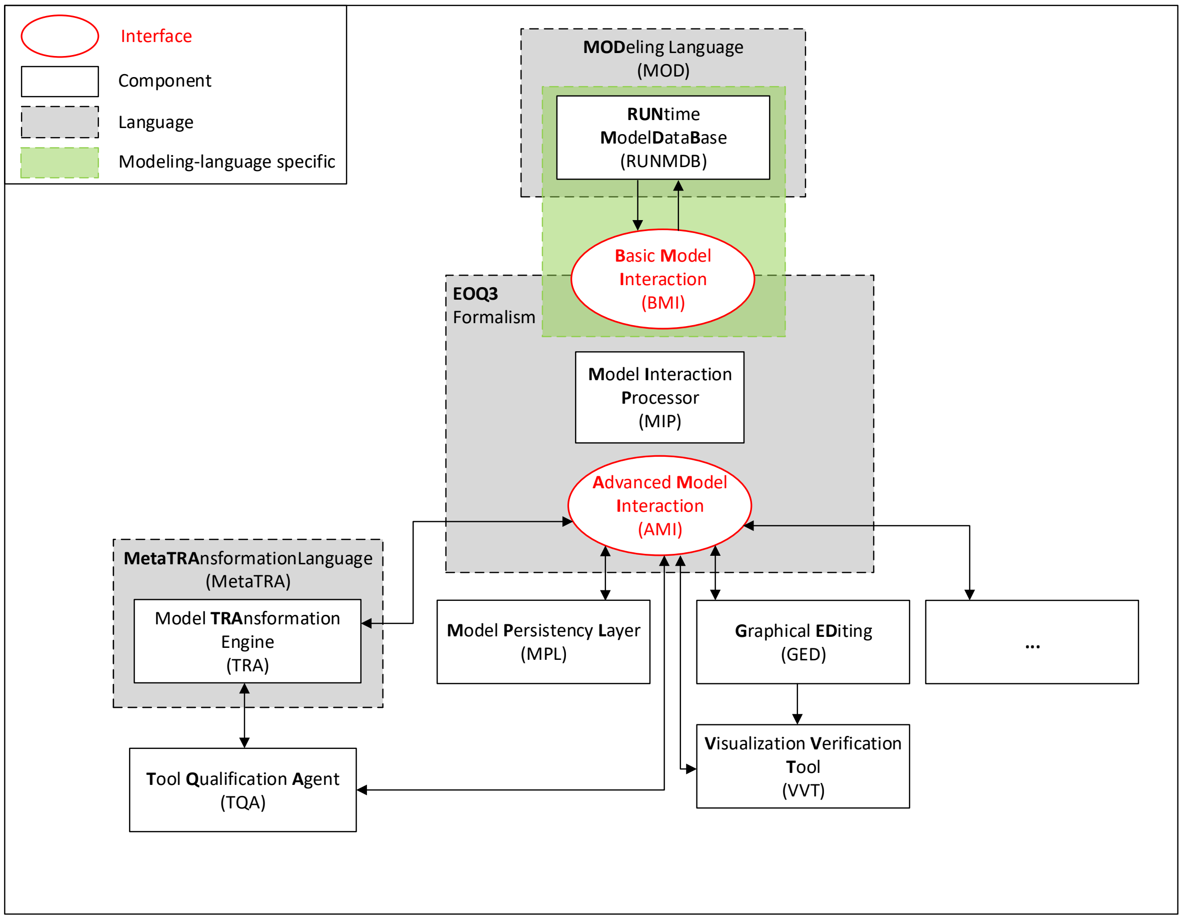
Interface
Component
Language
Modeling-language specific
MODeling Language (MOD)
RUNtime ModelDataBase (RUNMDB)
Basic Model Interaction (BMI)
EOQ3 Formalism
Model Interaction Processor (MIP)
Advanced Model Interaction (AMI)
MetaTRAnsformationLanguage (MetaTRA)
Model TRAnsformation Engine (TRA)
Model Persistency Layer (MPL)
Graphical EDiting (GED)
...
Tool Qualification Agent (TQA)
Visualization Verification Tool (VVT)

Fig. 4 Overview DOMAINES framework

Second, measures for the effort reduction by newly developed supporting tools.

4.3 Effort Reduction by Methods and Technology

The following text validates the capability of DOMAINES to reduce the qualification effort of DSM software derived from it.

Firstly considering **Measure No. 1**: Implementation with a programming language suitable for safety-critical programming. For this, we utilize the programming language Ada for the implementation of RUNMDB and MIP. Moreover, the key functionality of RUNMDB (creating, updating, reading, and deleting of model elements) shall later be restricted to Ada SPARK. SPARK is able to perform formal proofs and guarantees freedom from runtime errors, this could result in the fact that this proof does not have to be additionally executed and provided. Despite the possibility of conducting object-oriented programming with Ada, we decided to avoid it and use the procedural programming paradigm. The management of objects within RUNMDB is performed by using object-relational mapping where dependencies between objects are resolved by lists and references.

Decoupling and modularity are addressed by **Measure No. 2**. We want to achieve that components with a lower impact on aircraft safety can be certified with lower DAL or TQL and thus with less effort. E.g., the GED likely has a higher software complexity and lesser impact on aviation safety than the RUNMDB and therefore it is favorable to have less stringent qualification objectives. It should also be possible to remove individual components of the framework or use some of them in other environments, e.g. one may decide to use a different, possibly more convenient GED or someone wants to add a new functionality to the framework. Moreover, already qualified components could be used, such as qualified model-checkers or compilers. The straightforward interchangeability of components is enabled by AMI and BMI. The AMI interface is defined by an Extended Backus-Naur Form (EBNF) based grammar. As long as every input command corresponds to this defined grammar, it is possible to communicate with the MIP and the underlying RUNMDB. In contrast, the BMI interface is defined based on CRUD commands. Each request or query is resolved to basic CRUDs within the MIP. If the RUNMDB is changed, only the further processing of the CRUDs within the BMI must be adapted. The aim is that AMI and BMI are formally defined and all possible variants of model interaction can be tested. Modularity additionally plays a role in the TRA engine component. As depicted in Figure 5, **TRA** is divided into a *Transformation Executor*, a *Transformation Generator*, and the *Transformation Model* itself based on the *MetaTRAnsformationLanguage (MetaTRA)*. Only the *Transformation Executor* is part of the DSM tool to be qualified or certified. The *Transformation Generator* itself does not need to be qualified. The *Transformation Executor* is the output of the *Transformation Generator* for a specific transformation model. When the *Transformation Executor* is verified, this is equivalent to an output verification for the *Transformation Generator*. Thus, only the *Transformation Executor* has to undergo further qualification or certification activities.

As a further point, **Measure No. 3** is applied: provide a tool qualification kit to tool users. The deployment of a tool qualification kit is the purpose of the TQA. For the efficient and reliable generation and deployment, M2T transformations of TRA shall be used. The following documents, for example, should later be generated: Requirements documents including tool high-level and tool low-level requirements, testing results from tool testing against the tool requirements and traceability data for the traceability between tool requirements, tool implementation and tool testing [23].

Measure No. 4 - Application of restrictions, is addressed through a restrictive and simplified meta-modeling language (MOD), the use of text-based constraints, the application of the multi-level modeling concept, the usage of an Extended-Backus Naur Form grammar for command definition [56] [55], and restrictive rule scheduling. Our meta-modeling language, shown in Figure 6, is based on simplicity, which means that compared to other (meta)-modeling languages, functionality has been reduced to the one which is required from the described use cases to facilitate the certification process. This finally leads to a restriction of the modeling possibilities. MOD allows to create *Classes*, *Attributes*, and *Units* as well as to connect *Classes* with each other via *Compositions*, *Inheritances* and *Associations*. In addition, models can be structured using *Namespaces* and further restrictions can be created using *Mathematical Constraints* and *String-based* constraints. The elements of this language are limited to the 8 different elements. Each of these is itself a *StructureElement* which is more or less similar to EObjects from the Ecore meta-language. The *StructureElement* is required for the interchangeability of instances on M1 level. Due to Req5, it is not necessary to be able to model operations. Therefore, operations are not part of the meta-language [56]. In addition to the restricted meta-modeling language itself, we apply concepts of multi-level modeling [7]. With the introduction of potency values (@), it is possible to restrict the allocation of values to specific modeling levels. For example, we already want to specify at the M3 level, that attributes should be given a value at the M1 level instead of only having influence from one modeling level to the one above (M3 -> M2, M2 -> M1). This is denoted by the '@2' symbol behind the value attribute. This allows us to use the M3 language definition to influence and restrict M1 models to be created. Another example is that we want to achieve that each attribute is already assigned a data type when the domain-specific language is created (M2 level) and this cannot be changed at M1 level denoted by the '@1' symbol. As is depicted in Figure 6, we additionally provide the possibility to define constraints to restrict modeling possibilities on M1 level. Constraints are divided into mathematical constraints and string-based constraints and are formally defined by an Extended-Backus Naur Form (EBNF) grammar which is itself strongly

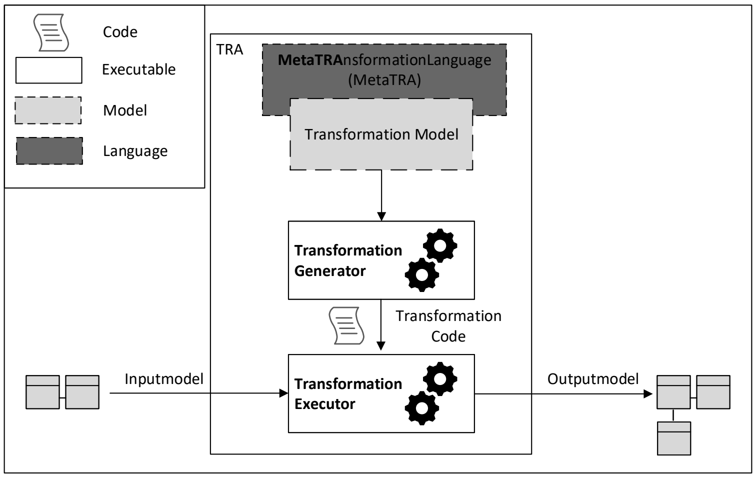

Fig. 5 TRAnsformation engine overview

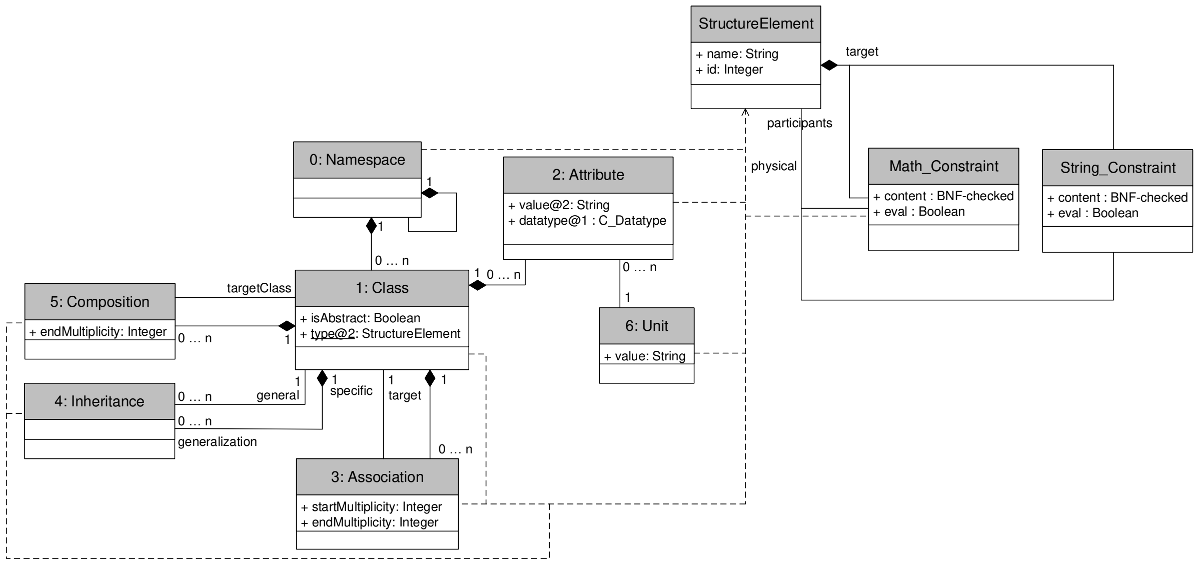

Fig. 6 DOMAINES meta-modeling language MOD [56]

restrictive [55]. DOMAINES uses a restrictive rule scheduling to execute transformations. The goal of restrictive rule scheduling is to ensure that each rule has a well-defined preceding and succeeding rule. This enables deterministic rule sequencing according to our properties for determinism presented in Section 3. The rule sequencing corresponds to a restrictive state diagram. Each rule corresponds to a node in the graph of the state diagram. The transformation begins at the start node and ends at the defined end node. The transition from one rule to the successor rule is defined by an edge between the rule nodes. Parallel execution paths are not permitted by the restrictions.

The usage of formal methods to verify the correctness of implementations is addressed in **Measure No. 5**. We aim for an implementation that conforms to SPARK, at least with the core functionality of creating and updating models. The definition of the transformation language is based on the formal methods of graph rewriting. In the specification, a transformation rule is divided into left-hand side patterns and right-hand side patterns. The execution of the rule is specified by a double pushout DPO-approach [19]. This approach enables the formal verification of the correct implementation of a transformation rule.

Measure No. 7 is a measure that applies to every component of the DOMAINS framework is the explicit decision to use **algorithms and implementation methodologies that are as simple and comprehensible as possible** in order to ease qualification and certification processes. This includes, for instance, the utilization of the non-recursive descent parser technology (LL(k)) instead of more complex, powerful, and faster bottom-up LR parsing algorithms. One huge advantage of an LL(k) parser is that the algorithm is comprehensible and does not require any pointers. Moreover, the algorithm is highly comprehensible. The implementation of RUNMDB makes extensive use of an advantage of the Ada programming language - the application of self-defined and comprehensible data types. This should increase the comprehensibility of the code and thus reduce the susceptibility to errors. In addition, this is intended to avoid errors caused by the use of incorrect data types.

4.4 Effort Reduction by Supporting Tools

During our work on the certifiable DSM framework DOMAINES, we have identified additional ways to support the reduction of the qualification effort.

Creating the necessary qualification artifacts for a DSM tool in the tool qualification process is associated with an immense manual effort for the tool user. For this reason, DOMAINES has a software component that supports the tool user in creating these artifacts, the **Tool Qualification Agent (TQA)**. It enables a model-based and automated generation of the required documents. The information for this is read from the models of the DSM tool, e.g. domain-specific model, meta-model, and transformation models. The transformation models, for example, contain information about the configuration, the scheduling and the individual rules of the model transformation. From this information, specifically developed model transformations for tool qualification automatically generate qualification documents. The model-based and automated approach of the

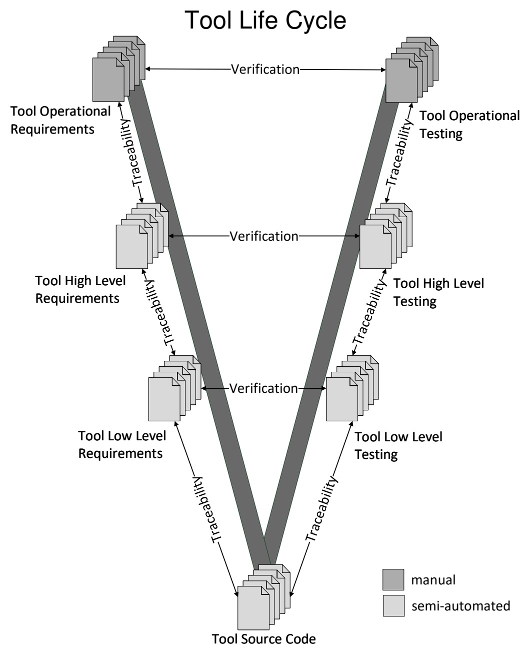

Fig. 7 Tool life cycle

Tool Qualification Agent should reduce the manual effort by an amount that makes the creation and use of custom DSM frameworks in avionics attractive.

The tool life cycle, which is supported by the Tool Qualification Agent, corresponds to a V-process, as depicted in Figure 7. At the operational level of the tool life cycle, the tool user is responsible for the manual creation of the operational requirements and the operational tool testing. The operational requirements are too abstract to be generated automatically by the Tool Qualification Agent, because the information is not stored in the models of the DSM tool. In the tool development process, the Tool Qualification Agent should generate the functional requirements and design requirements automatically. In the verification process, the Tool Qualification Agent should automatically generate the test documents (including test results). In addition, the Tool Qualification Agent is responsible for the traceability between the requirements and the tool testing. However, all created qualification artifacts must be manually reviewed by the tool user, as specified in the DO-330 standard.

Another point that, in our opinion, contributes to the goal of a DSM framework that can be used in the safety-critical avionics domain is the introduction of a **visualization verification tool (VVT)**. In an automated and tool-based development process, the developer needs to trust the visualization of object-based models. It is not convenient to verify in a poorly human-readable persistency file that the visualized and the stored models are identical. A typical problem might be that the developer deletes a modeling element, but for some reason, this change is only visualized and not propagated to the MDB. Another example originates from the human factors domain. What if the developer accidentally moves a block far away from the center of the model and forgets about its existence, however, the block is still there. In both cases, the stored model differs from the intention of the developer and thus leads to unintended behavior. Especially in the safety-critical domain, this needs to be avoided. The visualization per se does not ensure that displayed models match the data in the RUNMDB. This would require to qualify the visualization as a development tool with TQL-1 and respective effort. Rather a separate verification tool for the visualization is used as depicted in Figure 8. This allows qualification according to TQL-5, which means highly reduced qualification effort.

Measures from Table 2 taken for our approach to reduce the qualification or certification effort are summarized in the Table 3. It is important to note that these are primarily ideas and initial approaches. First prototypes already exist, however, still work needs to be done to quantify the benefits of the measures taken in DOMAINES towards simplified certification of DSM software.

4.5 Current Status of the DOMAINES Implementation

To validate the DOMAINES concept, a prototype was and is being developed. This prototype is available in the following git-repository: https://gitlab.com/domaines/domaines-main.

The current functionality of the prototype is to send strings or files with strings as AMI commands via the MIP and the BMI to the RUNMDB using a command line web-interface. Meta-models and domain-specific models can be created and modified in the RUNMDB. In addition, model information can be queried and elements can be deleted. With a read visualization, based on PlantUML, created models can be displayed. Instructions for creating an example avionics use case are available in the repository. This example involves the creation of the meta-model (M2) depicted in Figure 9. The M2 model allows for the creation of *Devices* which could be either *Devices* in general or specifications such as *Switches*, *Sensors*, or *Actuators*. Every *Device* or specialization of a *Device* contains *IOs* (Input/Output Interfaces). *IOs* are connected with each other via *Connections*. Each object has attributes such as *Size* and *Power* of a *Device*.

Based on this M2 meta-model it is possible to create a corresponding M1 domain-specific model. Due to the model size, the M1 domain-specific model is not suitable to be shown here. Please refer to the visualization in the prototype. The example from the manual includes the creation of four *Sensors*, two *Input-Output Modules (IOM)*, a *Switch*, a *Management Module*, an *Actuator Control*, and two *Actuators*.

Beside the creation of M2 and M1 models, the prototype can demonstrate methods to prevent possible misbehavior of the implementation or the user. The manual of the prototype gives two different scenarios:

- Send a command to the AMI which is not covered by the defined grammar. Wrong commands are rejected so that no misbehavior can be triggered within the RUNMDB.
- Send a command to the AMI with references to objects which are not available. Commands addressing objects not available within the model will be rejected to prevent unexpected behavior.

The prototype is continuously being developed. The next step is the integration of TRA. This should then make it possible to perform basic M2M transformations within the prototype.

After problems, proposed solutions and the own approach have been described in detail, the classification and delimitation to existing research and projects follows.

5 Related Work

DOMAINES is not the first approach that aims for a qualified use of DSM. In the follwing we list related work. As a teaser it can however be stated that there is no approach describing such a

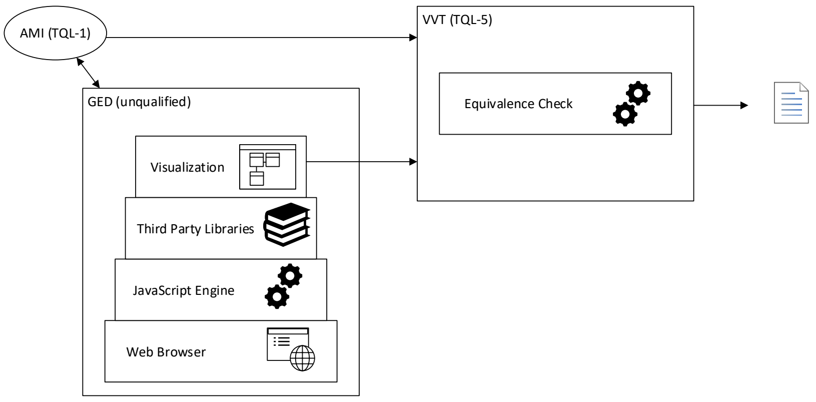

Fig. 8 DOMAINES Visualization Verification Tool in interaction with the Graphical Editor [60]

Table 3 Overview of Planed and Implemented Measures

	Measure	**Implementation**
1	Implementation with a programming language suitable for safety-critical programming and avoidance of OOP	Implementation with Ada (SPARK) and object relational mapping
2	Decoupling and modularity	Decoupling of components via AMI and BMI for different levels of qualification and certification effort, decoupled transformation generator and executor
3	Provide a tool qualification kit to tool users	Model-based and automated generation of requirements, test documents, and traceability data
4	Model restrictions	Restricted modeling possibilities due to simplified modeling language and concepts of multi-level modeling, design of an EBNF based constraint language, restrictive rule scheduling
5	Visual Verification Tool (VVT)	Ensuring that the visualization of the models corresponds to the data in the RUNMDB.
6	Tool Qualification Agent (TQA)	Responsible for qualification issues based on M2T transformations
7	Simple and comprehensible algorithms and implementations	LL(k) parsers, avoidance of pointers

comprehensive redesign of a DSM framework for certification as DOMAINES.

Within the scope of the Automotive Industry Working Group of the Eclipse Foundation, there was an approach to enable tool developers to use Eclipse as a platform for the development of qualifiable plugins within WP5 [21]. However, as far as can be seen from the available information, TQL-1 is excluded due to its high complexity. In addition, it is not clear whether this project was

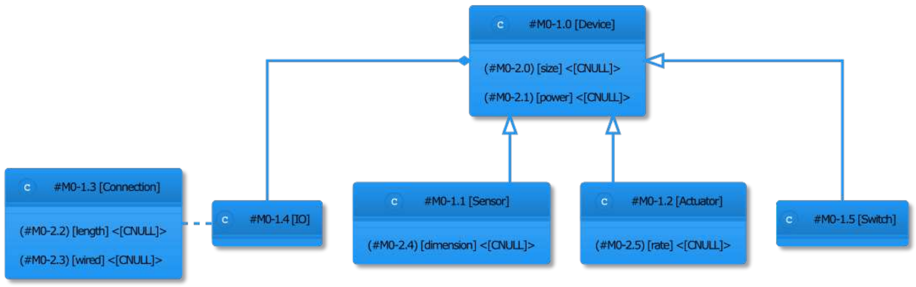
c #M0-1.0 [Device]
(#M0-2.0) [size] <[CNULL]>
(#M0-2.1) [power] <[CNULL]>
c #M0-1.3 [Connection]
(#M0-2.2) [length] <[CNULL]>
(#M0-2.3) [wired] <[CNULL]>
c #M0-1.4 [IO]
c #M0-1.1 [Sensor]
(#M0-2.4) [dimension] <[CNULL]>
c #M0-1.2 [Actuator]
(#M0-2.5) [rate] <[CNULL]>
c #M0-1.5 [Switch]

Fig. 9 M2 domain-specific language of prototype

pursued further after the concept phase. Similarly, Matlab Simulink claims to have a Tool Qualification Kit for the DO-178C standard. However, more detailed information is not publically available. We are not aware of any other approaches to make a framework of domain-specific modeling or parts of it qualifiable or certifiable.

For specific domains qualified approaches exist. Mainly certified code generators for dedicated models exist. Probably the most prominent example is SCADE [17]. SCADE is a model-based development environment for safety-critical embedded software. The code generator from SCADE models has been qualified as a development tool for DO-178B software up to Level A and DO-178C/DO-330 at TQL-1 [17]. Another qualified code-generator is the Gene-Auto [58], which is a qualified C code generator from mathematical models under Matlab-Simulink and Scilab-Scicos. TargetLink from dSpace [18] is a code generator for C code from Mathworks Simulink /Stateflow and is certified to ISO26262, ISO 25119, and IEC 61508 standards. Equally as TargetLink, the qualifiable code generation and verification tool QGen [1] is based on Simulink and Stateflow models. It is able to generate MISRA C code as well as SPARK (Ada subset) code which makes this generator suitable for DO-178C software. Moreover, when generating SPARK code, the code can be formally analyzed.

Approaches to at least reduce the qualification or certification effort exist in the form of model checkers or code analyzers. In [59] a case study was conducted in providing a qualification package for the Kind 2 model checker [13]. Polyspace from Mathworks [36] is a static code analysis tool, which is able to find bugs and formally proves the absence of critical runtime errors of source code from C, C++, and Ada programming languages. Moreover, Polyspace provides a DO178C qualification kit [59]. Other model checkers include UPPAAL, which can check invariant and reachability properties [10] or NuSMV [14] which is an open-source symbolic model-checker.

Last, there is an approach to support qualification administratively, by including certification information in development models [54] [33]. The IEC 61508 standard is integrated into a UML profile in order to enrich software models with certification information. Moreover, this profile is compliant with this standard.

6 Conclusion and Outlook

With this paper, we showed which difficulties arise when using DSM in (safety-critical) systems and software. In principle, DSM offers many advantages for the avionics domain, but existing tools and software can only be used in areas not liable to qualification itself or only to a very limited extend and with great effort as qualified tool or certified software. A distinction is made in the areas of application between development and practical use in the aircraft. We believe that there is currently no way to apply DSM in the aircraft at runtime except a manual verification of generated implementations. DSM is already being used for (safety-critical) system development. Nevertheless, we are of the opinion that the effort to make generated development artifacts usable is immensely high. The reasons for the high effort were analyzed and measures for mitigation were made. Moreover, we introduced our DSM framework DOMAINES which is currently under development and is designed to be qualifiable and certifiable. In doing so, we address which of the measures are and in the future be applied to reduce the qualification and certification effort. We see the introduction of a Tool Qualification Agent (TQA), the implementation with Ada (SPARK), the deterministic behavior of transformations, and the Visualization Verification Tool (VVT) as key points. More research has, however, to be conducted to quantify the effort reduction of a possible qualification or certification. The completion, validation and quantification of DOMAINES is subject to future research.

Declarations

Competing Interests and Funding The German Federal Ministry for Economic Affairs and Climate Action (BMWK) has funded this research within the LUFO-VI program and the TALIA project.

Availability of data and materials The datasets generated during and/or analyzed during the current study are available in the DOMAINES repository, https://gitlab.com/domaines.

References

[1] AdaCore (2022) QGen - the code generator you can trust. https://www.adacore.com/qgen, accessed: 2022-12-20

[2] Amrani M, Dingel J, Lambers L, et al (2012) Towards a model transformation intent catalog. p 3–8, https://doi.org/10.1145/2432497.2432499

[3] Annighoefer B (2019) An open source domain-specific avionics system architecture model for the design phase and self-organizing avionics. In: SAE Technical Paper Series, https://doi.org/10.4271/2019-01-1383

[4] Annighoefer B, Brunner M, Schoepf J, et al (2020) Holistic IMA platform configuration using web-technologies and a domain-specific model query language. In: 2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC), https://doi.org/10.1109/dasc50938.2020.9256726

[5] Annighoefer B, Brunner M, Luettig B, et al (2021) EOQ: An open source interface for a more DAMMMMN domain-specific model utilization. In: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, pp 483–492, https://doi.org/10.1109/MODELS-C53483.2021.00075

[6] Annighoefer B, Reinhart J, Brunner M, et al (2021) The concept of an autonomic avionics platform and the resulting software engineering challenges. In: 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). IEEE, https://doi.org/10.1109/seams51251.2021.00031

[7] Atkinson C, Kühne T (2001) The essence of multilevel metamodeling. In: International Conference on the Unified Modeling Language, Springer, pp 19–33, https://doi.org/10.1007/3-540-45441-1_3

[8] Balasubramanian D, Narayanan A, van Buskirk CP, et al (2006) The graph rewriting and transformation language: GReAT. Electron Commun Eur Assoc Softw Sci Technol 1. https://doi.org/10.14279/tuj.eceasst.1.89

[9] Batista L, Hammami O (2016) Capella based system engineering modelling and multi-objective optimization of avionics systems. In: 2016 IEEE International Symposium on Systems Engineering (ISSE), pp 1–8, https://doi.org/10.1109/SysEng.2016.7753127

[10] Behrmann G, David A, Larsen KG (2004) A tutorial on UPPAAL. Formal methods for the

design of real-time systems pp 200–236. https://doi.org/10.1007/978-3-540-30080-9_7

[11] Belschner T, Müller P, Reichel R (2019) Automated requirements and traceability generation for a distributed avionics platform. SAE International Journal of Advances and Current Practices in Mobility 1(2):447–457. https://doi.org/https://doi.org/10.4271/2019-01-1384

[12] Burns A (1999) The ravenscar profile. ACM SIGAda Ada Letters XIX(4):49–52. https://doi.org/10.1145/340396.340450

[13] Champion A, Mebsout A, Sticksel C, et al (2016) The kind 2 model checker. In: International Conference on Computer Aided Verification, Springer, pp 510–517, https://doi.org/10.1007/978-3-319-41540-6_29

[14] Cimatti A, Clarke E, Giunchiglia F, et al (1999) NuSMV: A new symbolic model verifier. In: International conference on computer aided verification, Springer, pp 495–499

[15] Dassault Systems (2022) Cameo systems modeler. URL https://www.3ds.com/products-services/catia/products/no-magic/cameo-systems-modeler

[16] Di Ruscio D (2007) Specification of model transformation and weaving in model driven engineering. dissertation, Dipartimento di Informatica Universita di L'Aquila

[17] Dormoy FX (2008) SCADE 6 a model based solution for safety critical software development. In: Embedded Real Time Software and Systems (ERTS2008)

[18] dSpace (2022) Targetlink dspace. https://www.dspace.com/de/gmb/home/products/sw/pcgs/targetlink.cfm#176_25806, accessed: 2022-01-25

[19] Ehrig H, Ehrig K, Prange U, et al (2010) Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. An EATCS Series, Springer Berlin Heidelberg

[20] Feiler PH, Gluch DP (2012) Model-Based Engineering with AADL. Addison-Wesley, Upper Saddle River, N.J

[21] Foundation E (2023) Auto iwg wp5. https://wiki.eclipse.org/Auto_IWG_WP5, accessed: 2023-01-04

[22] France R, Rumpe B (2005) Domain specific modeling. Software & Systems Modeling 4(1):1–3. https://doi.org/10.1007/s10270-005-0078-1

[23] Frey C, Annighoefer B (2023) Model-based and automated software tool qualification of domain-specific modeling tools as per RTCA DO-330, unpublished

[24] Halle M, Thielecke F (2015) Next generation IMA configuration engineering-from architecture to application. In: 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC), IEEE, pp 6B2–1, https://doi.org/10.1109/DASC.2015.7311445

[25] Halle M, Thielecke F (2016) Model-based transition of IMA architecture into configuration data. In: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), pp 1–10, https://doi.org/10.1109/DASC.2016.7777950

[26] Halle M, Thielecke F (2019) Tool chain for avionics design, development, integration and test. In: Software Engineering, https://doi.org/10.15480/882.2598

[27] Hilderman V (2014) DO-178C costs versus benefits. https://afuzion.com/do-178c-costs-versus-benefits/, accessed: 2022-02-15

[28] Ibrahim M, Durak U (2021) State of the art in software tool qualification with DO-330: A survey. Proceedings of the Software Engineering pp 22–26

[29] Institute for Software and Integrated Systems - Vanderbilt University (2022) MetaGME. https://www.isis.vanderbilt.edu/node/3924, accessed: 2022-02-15

[30] Iqbal MZ, Sartaj H, Khan MU, et al (2019) A model-based testing approach for cockpit display systems of avionics. In: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE, pp 67–77, https://doi.org/10.1109/MODELS.2019.00-14

[31] Jouault F, Kurtev I (2006) Transforming models with ATL. In: Satellite Events at the MoDELS 2005 Conference: MoDELS 2005 International Workshops Doctoral Symposium, Educators Symposium Montego Bay, Jamaica, October 2-7, 2005 Revised Selected Papers 8, Springer, pp 128–138, https://doi.org/10.1007/11663430_14

[32] Kärnä J, Tolvanen JP, Kelly S (2009) Evaluating the use of domain-specific modeling in practice. In: Proceedings of the 9th OOPSLA workshop on Domain-Specific Modeling

[33] Kuschnerus D, Bruns F, Bilgic A, et al (2012) A UML profile for the development of IEC 61508 compliant embedded software. In: Embedded Real Time Software and Systems (ERTS2012)

[34] Lasnier G, Zalila B, Pautet L, et al (2009) Ocarina : An environment for AADL models analysis and automatic code generation for high integrity applications. In: Reliable Software Technologies – Ada-Europe 2009. Springer, p 237–250, https://doi.org/10.1007/978-3-642-01924-1_17

[35] Ledeczi A, Maroti M, Bakay A, et al (2001) The generic modeling environment. Workshop on Intelligent Signal Processing, Budapest, Hungary 17

[36] MathWorks (2022) Polyspace making critical code safe and secure. https://www.mathworks.com/products/polyspace.html, accessed: 2022-01-25

[37] MathWorks Inc. (2022) System composer. https://www.mathworks.com/products/system-composer.html, accessed: 2022-02-15

[38] MIRA-Limited (2004) MISRA-C:2004 - Guidelines for the use of the C language in critical systems. MIRA Limited

[39] Mueller P, Belschner T, Lehmann M, et al (2018) AAA process: a new approach to affordable fly-by-wire systems for CS23 aircraft. CEAS Aeronautical Journal 9. https://doi.org/10.1007/s13272-018-0282-7

[40] Nordmann A, Hochgeschwender N, Wrede S (2014) A survey on domain-specific languages in robotics. In: Simulation, Modeling, and Programming for Autonomous Robots. Springer, p 195–206, https://doi.org/10.1007/978-3-319-11900-7_17

[41] Object Management Group (2016) MOF Query/View/Transformation. Standard ormal/2016-06-03, Object Management Group, Milford, USA, URL https://www.omg.org/spec/QVT/1.3

[42] Object Management Group (2017) OMG Unified Modeling Language. Standard formal/2015-03-01, Object Management Group, Milford, USA, URL https://www.omg.org/spec/UML/2.5/PDF

[43] Object Management Group (2019) OMG Meta Object Facility (MOF) Core Specification. Standard formal/2019-10-01, Object Management Group, Milford, USA, URL https://www.omg.org/spec/MOF/2.5.1/PDF

[44] Rierson L (2013) Developing Safety - Critical Software - A Practical Guide for Aviation Software and DO-178C Compliance. CRC Press - Taylor & Francis Group

[45] Rohl JS (1968) A note on backus naur form. The Computer Journal 10(4):336–337

[46] RTCA (2011) DO-178C software considerations in airborne systems and equipment. Standard, RTCA, Washington, USA, URL https://rtca.org

[47] RTCA (2011) DO-330 software tool qualification considerations. Standard, RTCA, Washington, USA, URL https://rtca.org

[48] RTCA (2011) DO-331 model-based development and verification supplement to DO-178C and DO-278A. Standard, RTCA, Washington, USA, URL https://rtca.org

[49] RTCA (2011) DO-332 object-oriented technology and related techniques supplement to DO-178C and DO-278A. Standard, RTCA, Washington, USA, URL https://rtca.org

[50] RTCA (2011) DO-333 formal methods supplement to DO-178C and DO-278A. Standard, RTCA, Washington, USA, URL https://rtca.org

[51] SAE (2010) Guidelines for development of civil aircraft and systems. Standard, SAE, URL https://www.sae.org/standards/content/arp4754a/

[52] Schoepf J, Annighoefer B, Reichel R (2019) A meta-model and transformation schema for the automated generation of ICDs in an automated development process of IMA system functions. In: Proceedings of the 7th International Workshop on Aircraft System Technologies. Shaker

[53] Steinberg D, Budinsky F, Paternostro M, et al (2009) EMF Eclipse Modeling Framework. Addison-Wesley Professional

[54] Subbiah S, Nagaraj S (2003) Issues with object orientation in verifying safety-critical systems. In: Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2003., https://doi.org/10.1109/isorc.2003.1199241

[55] Tietz V, Annighoefer B (2022) A formally defined and formally provable ebnf-based constraint language for use in qualifiable software. In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, pp 862–871, https://doi.org/10.1145/3550356.3561552

[56] Tietz V, Schoepf J, Waldvogel A, et al (2021) A concept for a qualifiable (meta)-modeling framework deployable in systems and tools of safety-critical and cyber-physical environments. In: 2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE, pp 163–169, https://doi.org/10.1109/MODELS50736.2021.00025

[57] Tietz V, Frey C, Schoepf J, et al (2022) Why the use of domain-specific modeling in airworthy software requires new methods and how these might look like? In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, pp 627–632, https://doi.org/10.1145/3550356.3561536

[58] Toom A, Naks T, Pantel M, et al (2008) Gene-auto: an automatic code generator for a safe subset of simulink/stateflow and scicos. In: Embedded Real Time Software and Systems (ERTS2008)

[59] Wagner L, Mebsout A, Tinelli C, et al (2017) Qualification of a model checker for avionics software verification. In: NASA Formal Methods Symposium, Springer, pp 404–419, https://doi.org/10.1007/978-3-319-57288-8_29

[60] Waldvogel A (2022) Towards qualifiable graphical editing of complex domain-specific models in safety-critical avionics. In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, pp 158–163, https://doi.org/10.1145/3550356.3558507